CS193J: Programming in Java
Winter Quarter 2003

MVC/JTable, Exceptions and Files

Manu Kumar
sneaker@stanford.edu

Agenda

- Last Time:
  - Threading continued (wait/notify)
- Today:
  - CS193J Tips and Tricks
  - MVC
    - Model View Controller paradigm
    - JTable
  - Exceptions
  - Files and Streams

Handouts

- 4 Handouts for today!
  - #23: Homework #3 Part B
  - #24: MVC/Tables
  - #25: Exceptions
  - #26: Files and Streams
- Homework #3 (Part a and b) tips
  - This homework is *not* short
  - Start early
    - Threading/Concurrency bugs are the hardest bugs you will ever encounter!

While we're talking about tips…

- Some random tips to help you
  - Syntax highlighting in emacs
    - If you use emacs in an X environment, you can turn on syntax highlighting under the Options menu
  - VNC is your friend
    - Leland has the VNC Server installed on it already!
    - Download and installed VNC Client from http://www.realvnc.com
  - System.out.println() is your life-saver!
    - When debugging, always create utility methods to dump your object state and use System.out.println() to be able to view it
    - When using Threads –output the thread name (Thread.getName() method) so that you know which thread is active

Homework #3 Part b intuition

- How many of you have *not* used Napster/Kazaa/Bearshare! ☺
  - The interface HW3 presents for checking links is reminiscent of how P2P filesharing clients download files.

Link tester

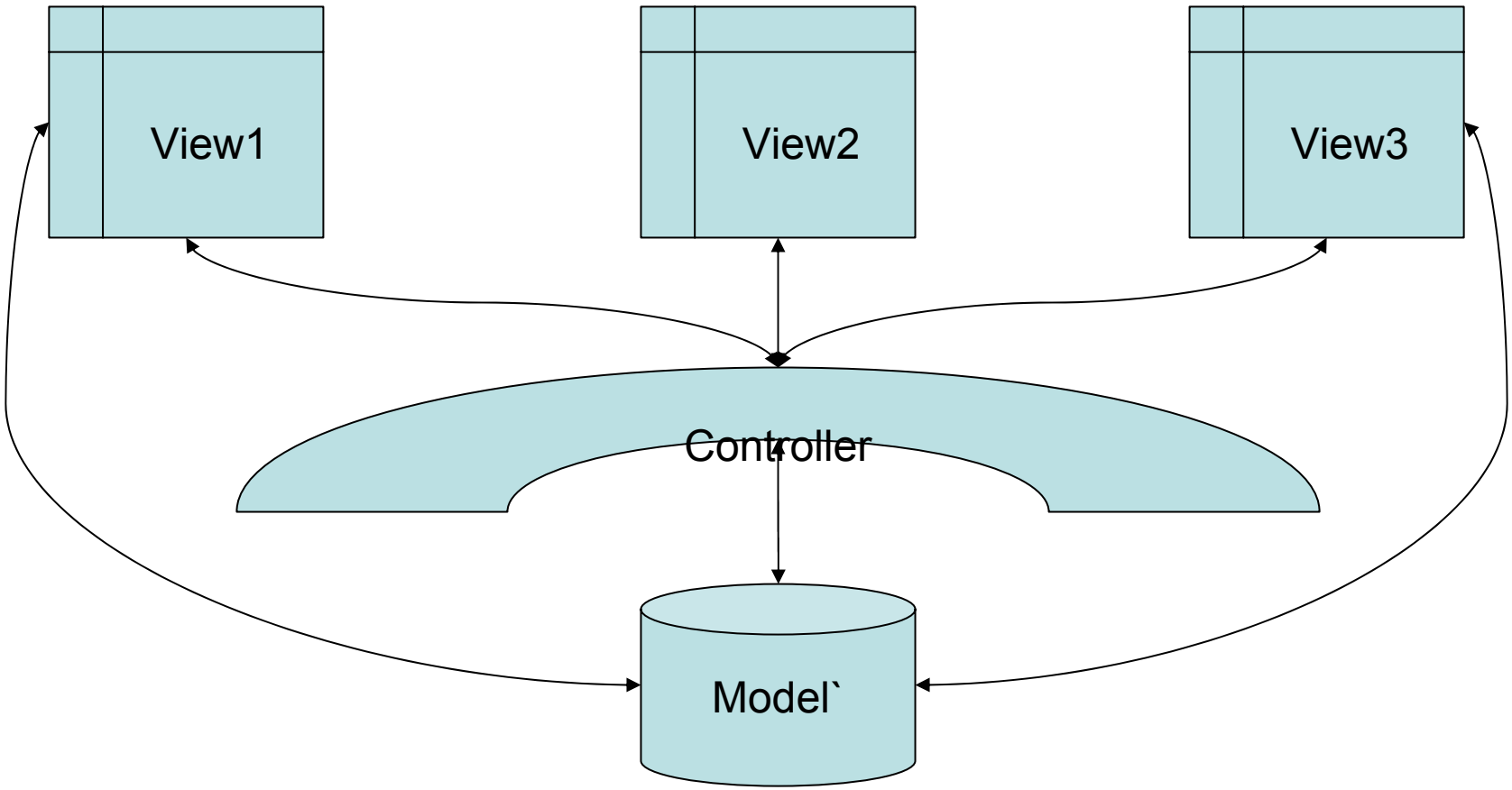
Start URL: http://www.stanford.edu/class/cs193j/materials/hw3/page2.html Max threads 5 Test links

#	Link	Type	Status	Downloaded	Time	Throughput	Progress
0	http://www.stanford.edu/class/cs193...	text/html	Done	1395 of 1395 bytes	3737 ms	373 bytes/sec	100% done
1	http://www.stanford.edu/	text/html	Reading	6144 of ??? bytes	8674 ms	708 bytes/sec	Indeterminate
2	http://www.stanford.edu/class/cs193...	text/html	Done	1922 of 1922 bytes	2674 ms	718 bytes/sec	100% done
3	page1.html	text/html	Done	641 of 641 bytes	1639 ms	391 bytes/sec	100% done
4	/class/cs108/	text/html	Done	1325 of 1325 bytes	2175 ms	609 bytes/sec	100% done
5	images/	text/html	Done	1910 of 1910 bytes	5229 ms	365 bytes/sec	100% done
6	gopher://dendrome.ucdavis.edu/	text/html	Done	1763 of 1763 bytes	5336 ms	330 bytes/sec	100% done
7	http://www.stanford.edu/class/cs193...	application...	Reading	3072 of 16539 bytes	5654 ms	543 bytes/sec	18% done
8	http://www.stanford.edu/class/cs193...	text/html	Reading	1024 of ??? bytes	1292 ms	792 bytes/sec	Indeterminate
9	ftp://ftp.stanford.edu/class/cs193j/a...		Connecting				
10	http://www.stanford.edu/class/cs193...		Pending				
11	file:/usr/class/cs193j/WWW/annou...		Pending				

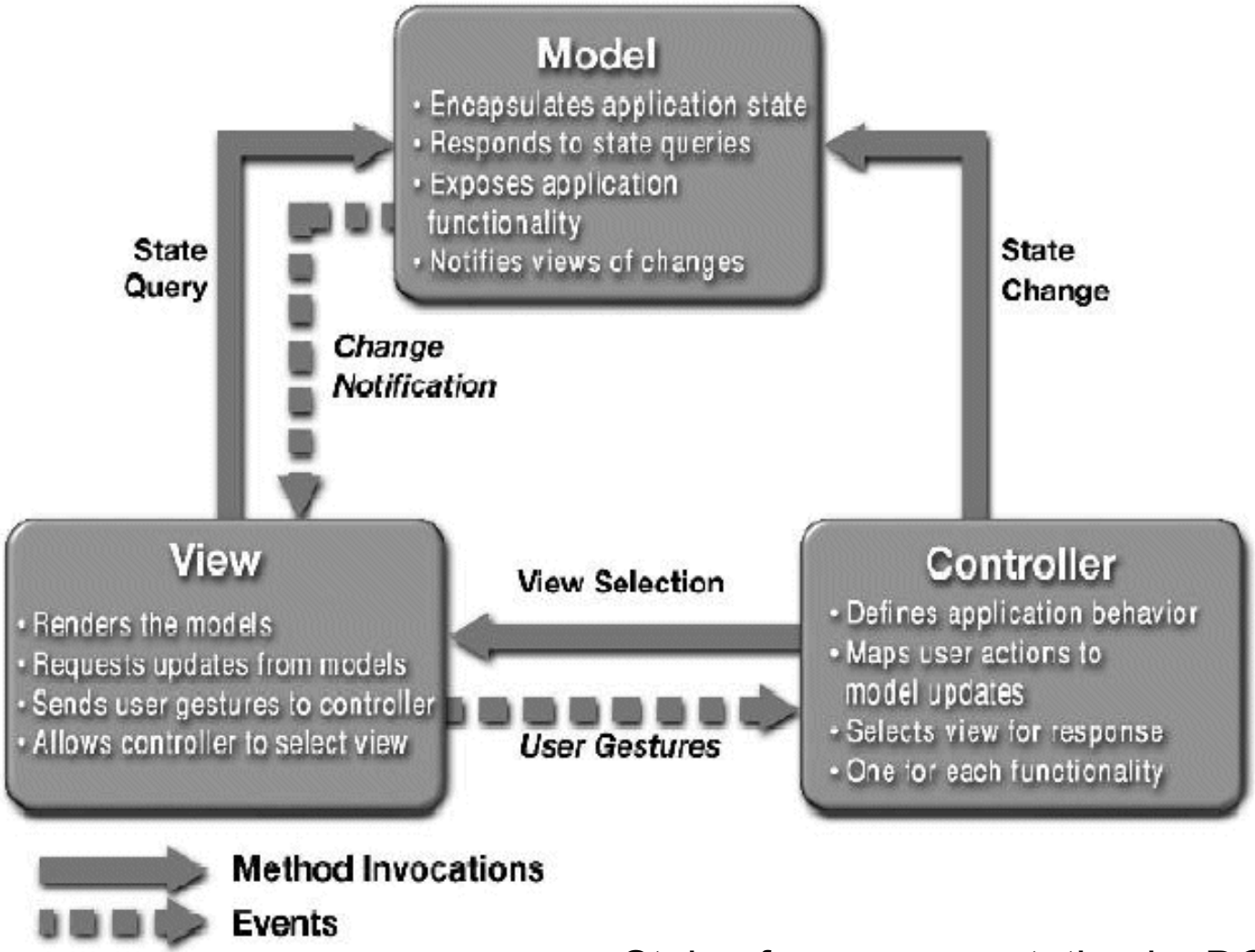
MVC

- MVC paradigm
  - **M**odel
    - Data storage, no presentation elements
  - **V**iew
    - No data storage, presentation elements
  - **C**ontroller
    - Glue to tie the Model and the view together
- Motivation
  - Provides for a good way to partition work and create a modular design
  - Very flexible paradigm for providing multiple ways to look at the same information

Rudimentary MVC diagram



Sun's MVC Pattern Diagram



Stolen from a presentation by DChen @ Sun

Tables in Swing

- Tables are one of the more involved UI elements in Swing
  - Basic functionality however it easy
  - Learn by pattern matching!
- Resources:
  - Handout has lots of sample code
    - Source for the code in the handout is available in electronic form on the course website
  - Sun's Java Tutorial on How to Use Tables
    - http://java.sun.com/docs/books/tutorial/uiswing/components/table.html

Tables in Swing

- Use MVC pattern!
  - Model: TableModel
  - View: JTable
  - Controller: UI elements and listener bindings
- JTable
  - Relies on a TableModel for storage
  - Has lots of features to display tabular data
- TableModel Interface
  - getValueAt(), setValueAt(), getRowCount(), getColumnCount() etc.
- TableModelListener Interface
  - tableChanged(TableModelEvent e)

AbstractTableModel

- Implements common functionality for TableModel Interface
  - But it is abstract, so you must extend it
    - getRowCount(), getColumnCount(), getValueAt()
  - Helper methods for things not directly related to storage
    - addTableModelListener(), fire___Changed()
- DefaultTableModel
  - Extends AbstractModel, but uses a Vector implementation

BasicTableModel

- Provided by Nick
  - Uses ArrayList implementation
  - getValueAt() to access data
  - setValueAt() to change data
    - Notifies of changes by sending fireTable____() methods
  - Handles listeners
- *This is what you should follow!*

Live Example!

TableFrame

Name	Favorite Thing
Barney	Saying please and thank you
Tinky Winky	Playing with my purse
Dr. Ross	Not Being on TV
Elvis	Drugs, etc.

Add Row
Add Column
Delete Row
Load File

Name	Favorite Thing
Barney	Saying please and thank you
Tinky Winky	Playing with my purse
Dr. Ross	Not Being on TV
Elvis	

Table Tips!

- Put the JTable in a JScrollPane
  - This automatically deals with handling space for the header and does the right things!
- To change column widths

```
TableColumn column = null;
for (int i = 0; i < 5; i++) {
    column = table.getColumnModel().getColumn(i);
    if (i == 2) {
      column.setPreferredWidth(100); //second column is bigger
    } else {
    column.setPreferredWidth(50);
    }
}
```

Exceptions

- You've seen these already!
  - So you already have some intuition about these
- Exceptions
  - Are for handling errors
  - Example:
    - ArrayIndexOutOfBoundsException
    - NullPointerExeption
    - CloneNotSupportedException

Error-Handling

- Programming has two main tasks
  - Do the main computation or task at hand
  - Handle exceptional (rare) failure conditions that may arise
- Bulletproofing
  - Term used to make sure your program can handle all kinds of error conditions
- Warning
  - Since error handling code is not executed very often, it is likely that it will have lots of errors in it!

Traditional Approach to Error Handling

- Main computation and error handling code are mixed together

```
int error = foo(a, &b)
If (error = 0) { ….}
```

- Problems
  - Spaghetti code – less readable
  - Error codes, values have to be manually passed back to calling methods so that the top level caller can do something graceful
  - Compiler does not provide any support for error handling

The Java Way: Exceptions

- Formalize and separate error handling from main code in a structured way
  - Compiler is aware of these “exceptions”
  - Easier to read since it is possible to look at main code, and look at error cases
  - Possible to pass errors gracefully up the calling hierarchy to be handled at the appropriate level

Exception Classes

- Throwable
  - Superclass for all exceptions
- Two main types of exceptions
  - Exception
    - This is something the caller/programmer should know about and handle
    - Must be declared in a *throws* clause
  - RuntimeException
    - Subclass of exception
    - Does not need to be declated in a *throws* clause
    - Usually reserved for things which the caller cannot do anything and therefore also usually fatal.

Exception Subclasses

- Exceptions are organized in a hierarchy
  - Subclasses are most specific
  - Higher level exceptions are less specific
- You can create your own subclasses of exceptions which are application specific
  - Rule of thumb: if your client code will need to distinguish a particular error and do something special, create a new exception subclass, otherwise, just use existing classes.

Methods with Exceptions

- Exception *throw*
  - *throw* can be used to signal an exception at runtime
- Method *throws*
  - When a method does something that can result in an error, it should declare *throws* in the method declaration

```
public void fileRead(String f) throws IOException {
....
}
```

"Handling" Exceptions

- Two possible options
  - Pass-the-buck-approach
    - Declare the exception in a *throws*
    - This passes the exception along to the caller to handle
  - Do-Something-approach
    - Use *try-catch* block to test if an exception can happen and then so something useful
- Which one to use:
  - Depends on the application!

try / catch

- Idea:
  - “try” to do something
  - If it fails “catch” the exception
  - Do something appropriate to deal with the error
- Note:
  - A *try* may have multiple *catch*es!
    - Depending upon the different types of exceptions that can be thrown by all the statements inside a try block
  - Exceptions are tested in the same order as the catch blocks
    - Important when dealing with exceptions that have a superclass-subclass relationship

try / catch example

```
public void fileRead(String fname) {                    // NOTE no throws

    try {
            // this is the standard way to read a text file...
            FileReader reader = new FileReader(new File(fname));
            BufferedReader in = new BufferedReader(reader);

            String line;
            while ((line = in.readLine()) != null) {
                        ...
            // readLine() etc. can fail in various ways with
            // an IOException                    }
    }
    // Control jumps to the catch clause on an exception
    catch (IOException e) {
            // a simple handling strategy -- see below for better strategies
            e.printStackTrace();
    }
}
```

printStackTrace() is your friend!

- When dealing with exceptions
- Especially when debugging
- printStackTrace() will:
  - Show you the full calling history
  - With line numbers
- Note:
  - *Bad* idea to eat an exception silently!
  - Either printStackTrace() or pass it along to be handled at a different level

Files and Streams

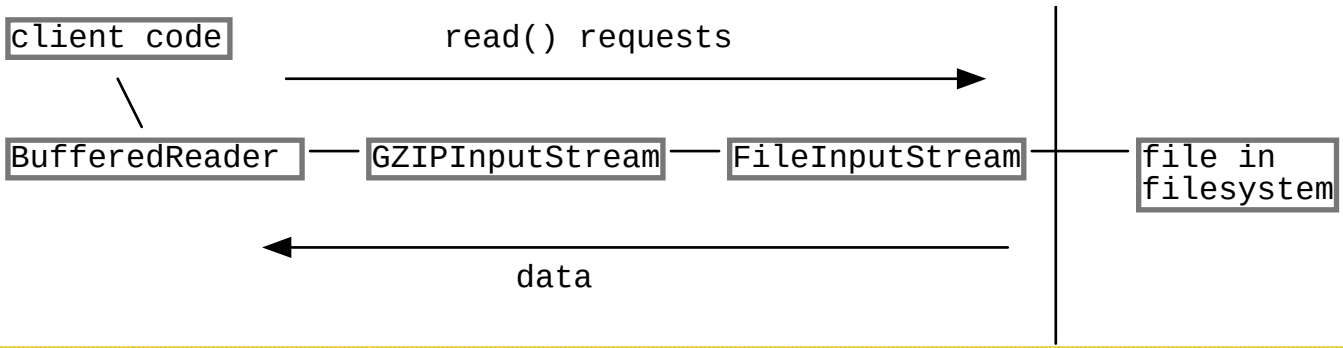
- File
  - Represents a file or directory
  - Java abstracts away the ugliness of dealing with files quite nicely
- Streams
  - Way to deal with input and output
  - A useful abstraction…

Streams!??

- Water analogy
  - Think of streams as pipes for water
  - Do you know whether the water that comes out of your tap is coming from a) the ocean b) some river c) a water tank d) a water buffalo?
- Idea:
  - You abstract away what the stream is connected to and perform all your I/O operations on the stream
  - The stream may be connected to a file on a floppy, a file on a hard disk, a network connection or may even just be in memory!

Hierarchy of Streams

- Java provides a hierarchy of streams
  - Think of this as different “filters” you can add on to your water pipe
    - Some may compress/decompress data
    - Some may provide buffers
- Common Use Scenario
  - Streams are used by layering them together to form the type of “pipe” we eventually want



Types of Streams

- InputStream / OutputStream
  - Base class streams with few features
  - read() and write()
- FileInputStream / FileOutputStream
  - Specifically for connecting to files
- ByteArrayInputStream / ByteArrayOutputStream
  - Use an in-memory array of bytes for storage!
- BufferedInputStream / BufferedOutputStream
  - Improve performance by adding buffers
  - Should almost always use buffers
- BufferedReader / BufferedWriter
  - Convert bytes to unicode Char and String data
  - Probably most useful for what we need

Streams and Threads

- When a thread sends a read() to a stream, if the data is not ready, the thread blocks in the call to read(). When the data is there, the thread unblocks and the call to read() returns
- The reading/writing code does not need to do anything special
- Read 10 things at once – create 10 threads!

Reading Example

```
public void readLines(String fname) {
    try {
        // Build a reader on the fname, (also works with File object)
        BufferedReader in = new BufferedReader(new
    FileReader(fname));
        String line;
        while ((line = in.readLine()) != null) {
            // do something with 'line'
            System.out.println(line);
        }

        in.close();          // polite
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
```

Writing Example

```
public void writeLines(String fname) {
    try {
        // Build a writer on the fname (also works on File objects)
        BufferedWriter out = new BufferedWriter(new FileWriter(fname));

        // Send out.print(), out.println() to write chars
        for (int i=0; i<data.size(); i++) {
            out.println( ... ith data string ... );
        }

        out.close();                    // polite
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
```

HTTP

- Java has build-in and very elegant support for HTTP
- Code on the handout is what you will need for HW #3 Part b!
- URL
  - Uniform Resource Location
    - http://cs193j.stanford.edu
- URLConnection
  - To open a network connection to a URL and be able to get a stream from it to read data!

HTTP Example

```
public static void dumpURL(String urlString) {
        try {
                URL url = new URL(urlString);
                URLConnection conn = url.openConnection();
                InputStream stream = conn.getInputStream();
                BufferedReader in = new BufferedReader( new
InputStreamReader(stream));

                String line;
                while ( (line = in.readLine()) != null) {
                        System.out.println(line);
                }
                in.close();
        }
        catch (MalformedURLException e) {
                e.printStackTrace();
        }
        catch (IOException e) {
                e.printStackTrace();
        }
}
```

Summary!

- Today
  - Tips and Tricks
  - MVC / Tables
  - Exceptions
  - Files and Streams
- Homework #3 Part b handed out!